Florida Tech

Project Report

The Marriage of Ada and Joe College: The Future is Now

MDA972-92-J-1004

Principal Investigator:

Charles B. Engle, Jr., Ph.D.
Florida Institute of Technology

Executive Summary

The Florida Institute of Technology has embarked on a unique experiment to develop an undergraduate curriculum in software engineering. The first year of this new program is devoted to programming-in-the-small issues, including traditional programming practices using the Ada programming language, small program design techniques, and program verification and debugging. In addition, the concepts of formal syntax and semantics, formal grammars and BNF, performance issues, abstractions, and object-oriented issues have been introduced. The course draws from the many knowledge units described in the ACM/IEEE-CS Computing Curricula 1991 Report [ACM 91].

The second year of the program, for which this grant provided funding support, concentrates on programming-in-the-large, software engineering concepts. The focus of this year is software systems development using teams of interacting persons, the use of tools, configuration management, verification and validation, testing, quality assurance, and more complex design techniques.

Two courses were created to provide maintenance and development experiences for teams of students. The first course concentrates on *maintenance* activities using an artifact and its associated development documentation. The second course concentrates on software systems *development* using Cleanroom technology.

The innovative portion of this approach is that:

- it uses Ada throughout all course activities.
- it approaches software development from a software engineering perspective using a team approach.
- it emphasizes Cleanroom technology.
- it requires the students to function in an industrial-style setting.
- it requires the students to use the controlling disciplines of configuration management, quality assurance, and verification and validation as part of the course.
- it uses an artifact as a basis for instruction, presenting the student with a motivation for the controlling disciplines not normally found in typical or traditional small student projects.
- it requires most of the knowledge units specified by the new ACM curriculum.

Unfortunately, the results of this experiment have not been as valuable as was hoped. It is unlikely that any other institution will find this material useful since it is so very much tailored to the unique program we have here at Florida Tech. Even within that program, we will need to make some reasonably major modifications to continue to use the material because of the lack of success obtained.

I. Introduction

The purpose of this research was to develop a sophomore level curriculum for software engineering using Ada. The sophomore level consists of three quarter courses at the Florida Institute of Technology (Florida Tech) and the curriculum was designed as three courses of classroom instruction augmented by laboratories. Since Florida Tech is converting to a semester system, these course were designed with a view toward easing the transition from the original three quarter courses to two semester courses reflecting the same course content. Our purpose was to specify, design, and develop a one year sequence of courses in software engineering using Ada throughout as the language of illustration. The development of these courses was scheduled to take place during the period 1 September 1991 through 13 June 1992, but the period was extended until August 1993 because of the need to refine our approach.

Specifically, Florida Tech :

- examined the issues in undergraduate software engineering and attempted to specify the learning objectives for second year students.
- designed a sequence of courses to meet these learning objectives.
- developed the course materials needed to implement this sequence of courses, including, but not limited to, learning objectives by lesson, lesson outlines, course outlines, lecture notes for each lecture, team exercises for the students, laboratory exercises, laboratory manuals, instructor's guides, sample examinations, overhead slides, and a rationale for the courses as implemented.
- served as a consultant for other instructors interested in using this course sequence.

II. Background

Sophomore courses have been taught at numerous institutions for many years. Courses targeted for computer science have similarly been taught for many years. However, software engineering is a new field and undergraduate course in this discipline have not been widely available. Because of the uniqueness of the program at Florida Tech, we were able to take advantage of the flexibility of our curriculum to tailor it specifically for software engineering.

The motivation to title a software systems course sequence as *engineering* is to focus on the fact that the courses have the structure, rigor, and formal foundation embodied in such a name. As with other engineering disciplines, the formal beginning must evolve from experiences of the initial practitioners [Shaw 90]. Over time fundamental constructs evolve and educational pioneers begin to put in place prototypical curricula. The implementation of these curricula provides the basis, ultimately, for an undergraduate degree in the field. During the past several years, we have seen the emergence of software engineering as a

discipline apart from computer science. This can be observed in the creation and publication of software engineering sample curricula in the ACM/IEEE-CS 1991 Computing Curricula [ACM 91], the British Computer Society and Institution of Electrical Engineers report on Undergraduate Curricula for Software Engineering [BCS 89], and the SEI Undergraduate Curriculum in Software Engineering [Ford 90]. Additionally, the Accreditation Board for Engineering and Technology (ABET) has recommended that all Computer Engineering majors take a course in software engineering; a recognition of the importance of this field to the computer engineering discipline, as well as an endorsement of the viability and uniqueness of software engineering apart from computer science.

This course sequence is founded on a firm theoretical foundation in which programs are viewed as rules for functions [Mills 91]. It embodies many of the ideas found in traditional engineering courses in terms of hands-on laboratories and student acquisition of practical applications of this theory in the lab. It is also predicated upon the notion that quality software can be developed assuming that the appropriate controls are in place during development and that usage testing under statistical quality control is performed. While much of this is language independent, the mechanism for demonstrating to the students the application of these ideas is the programming language Ada. Ada was chosen after careful consideration, and in spite of many difficult obstacles, because it is, quite simply, the best tool available for imparting the techniques of software engineering in the coding phase of the life cycle. The power of Ada used in a disciplined environment can be used to demonstrate in a tangible manner the concepts that we wish for our students to learn.

The impact of this approach is that students are better prepared to enter the field of software engineering. Many human resource managers from industry, especially the aerospace industry that deals mainly in government contracts, have told us that they would rather hire engineers from virtually any engineering discipline and train them in software development, than to hire today's computer science graduate. The reason is that the engineering students understand efficiency and tradeoffs, and are generally more disciplined, than their computer science peers. Further, it is our analysis that computer science is better suited for programming-in-the-small issues than for the programming-in-the-large issues that are typical for the large aerospace organizations. Thus, the time is right to transition the technology gained by experience to incoming students to better prepare them for the challenges of large-scale software systems development. We believe that students should leave behind the notion of *writing programs* to embrace the concept of *developing software systems*. While we do not want to be viewed as a training organization, we must nevertheless be pragmatic in our approach to education.

III. Procedure

We felt that the major emphasis in the new program implied moving from programming-in-the-small to programming-in-the-large. This meant that after having spent the freshman year learning about programming concepts at the single programmer level, with programs requiring the students to work independently, we felt that students needed to be brought together and be taught to work productively as part of a team. The sophomore year of the computer science curriculum at Florida Tech current consisted of three quarters, although the planning for these course sequences considered the fact that in 1993 Florida Tech changed to a semester program. Thus, one and one half quarters was the target of each of the courses described, so that the transition to semesters will be seamless. Therefore, the first course (actually one and one half quarters) was initially devoted to maintenance of large (to the students) software systems, while the second course (again, actually one and one half quarters) was devoted to software systems development.

The first course was designed to concentrate on maintenance. Specifically, the class was organized into two chief programmer teams as described by Brooks [Brooks 82]. Team concepts and organizational issues were presented. Students were then able to choose a role to play in the team effort using the team member positions described by Tomayko [Tomayko 87A, Tomayko 87B]. The teams were provided an artifact, selected from among those provided by the Software Engineering Institute (SEI), and all of the associated development documents consisting of many of the deliverables required under DOD-STD 2167A. This artifact was used to perform selected enhancements and corrections. This was accomplished by having the students form a change control board and providing the board a series of change requests and discrepancy reports. A budget was also provided to the team project administrator with cost codes for the various activities that the students performed and dollar amounts allocated for each cost code. The students then selected which change requests and discrepancy reports to accept based upon technical and budgetary considerations. They then had to perform the maintenance effort and deliver the product.

To implement this course design, we started the classroom instruction as a traditional software engineering overview, introducing life cycle models and discussing issues that pertain to life cycle phases. This was initially augmented in the laboratory by providing team building exercises. The Psychology Department at Florida Tech supplied us with some exercises that are traditionally used to foster team building. We had our students complete the exercises as part of their laboratory.

Classroom instruction meanwhile, continued with a detailed discussion of each phase of the life cycle. Issues in requirements gathering and analysis were examined and discussed, followed by specifications, design, implementation,

testing, and maintenance. Each topic was expanded to provide the student with a background to fulfill their portion of the laboratory exercises.

The laboratory exercises were designed, after the initial team building phase, to provide the students with as close to a "real world" experience as is possible in an academic setting. Toward that end, the maintenance exercises described in the Software Engineering Institute Educational Materials [Engle 89A] were provided as a laboratory artifact. The actual artifact is approximately 10,000 source lines of code in Ada. Exercises are described in that report on how to change the artifact to provide new functionality (perfective maintenance) and how to correct known deficiencies (corrective maintenance). In addition, the artifact as written to run on a VAX computer under the VMS operating system. Our students ran it on a SPARC using Unix. This meant that some environment changes had to be provided (adaptive maintenance). However, this was not the interesting part. What constituted a new approach was that the students were divided into two teams, each of which had positions that are normally found in software development (and thus maintenance) teams in software development organizations. Specifically, one student was chosen to be in the role of the task leader, another was the project administrator, another was the configuration manager, another was quality assurance, etc. The choice was open to the student who interviewed for the position desired. The interviews were videotaped (with the student's permission) and the student was provided with a critique of their interview as a guide for when they will be interviewing for "real" jobs. An approach modeled on one described by Dr. James Tomayko at the graduate level, was adopted (see "Teaching a Project-Intensive Introduction to Software Engineering" [Tomayko 87A, Tomayko 87B]).

After one and one half quarters, the students delivered an improved version of the original artifact. Delivery included reviews and inspections of both the design and the implementation.

The second course was designed to concentrate on software systems development. The students were again working in teams, but this time they did not have an artifact upon which to base their work. Instead, they were provided a requirements document containing the usual set of ambiguities associated with using the English language as a descriptor, from which they were to create a specification for the system to be developed. A Cleanroom Software Engineering approach was taught, with the student teams divided into three units each, consisting of a specification unit, a design and implementation unit, and a certification unit. The specifications created by the students were turned over to the design and implementation unit, that used box structured design techniques to create a design that was subjected to preliminary design reviews and critical design reviews with the whole class participating. Actual implementation of the system was not likely due to time constraints, but all software system development activities up to implementation were to be carried out by the students.

The implementation of this second half design of the year long curriculum included a new development project conducted under the Cleanroom approach to software development. In this portion of the course, the students developed a program to manipulate a robotic arm provided to them. Originally, the arm was to be connected to a port on the SPARC machine provided for their use. When the students were unable to manipulate the robot through the SPARC port, they decided to switch platforms and manipulate the arm through the port on a PC. This decision was made only after a careful analysis of their alternatives and the cost-tradeoffs involved. Although the robotic arm never did complete its assigned purpose, the learning experience of the students was very positive.

After the experiences of this first time teaching the course in this format, we evaluated our students to compare them against students taught using prior curricula. It was a subjective assessment, but the instructors unanimously felt that the students were strong in team cooperation skills and in their understanding of the controlling disciplines (configuration management, quality assurance, verification and validation, etc.), but they were too weak in programming and in data structures. Accordingly, we modified the entire program for the next year. This caused us to delay the submission of this report until the results of that modification were assessed.

In the second version of the curriculum for sophomores, we spent the first quarter and a half teaching the students fundamental data structures to correct the perceived deficiency from the first iteration. This built upon their earlier foundations in programming-in-the-small. The students then performed a software development exercise using a surgical team approach (as discussed in [Brooks 82], but with a smaller artifact. The results were very encouraging.

The first iteration used the SunAda compiler on the SPARC and the Meridian Ada compiler on the PC. The second iteration used solely the Meridian Ada compiler on the PC. Cadre Teamwork was obtained for use by the students, but was deemed too difficult to use by this level of student. Thus, only instructor demonstrations of its potential were provided for the students.

IV. Conclusions

Our first attempt to radically change the curriculum might be described as too much too soon. It had several deficiencies and was not appropriate for further dissemination. We then decided to modify this approach and restructure the curriculum to correct obvious short-comings. Our second iteration of this curriculum provides for a more traditional approach for the first semester (although out of the normal sequence for a data structures course), and the innovative portion is more or less restricted to the second semester. Of course, this presupposes a radically different first year, which our program continues to use. The results, in terms of lesson plans and course material, are probably not

usable at many other institutions. Since our program is so unique and the situations we encountered are so very different than traditional programs, this curriculum will not likely be of use to other institutions. Therefore, we will provide information on request to interested organizations, but will not disseminate this material widely. Our experiences have been disappointing, but the effort has not been wasted. Many lessons-learned were obtained that have already proved valuable and no doubt will continue to prove valuable in the future.

V. Bibliography

[ACM 91] ACM/IEEE-CS Joint Curriculum Task Force, "Computing Curricula 1991", Report of the ACM/IEEE-CS Joint Curriculum Task Force, 1991.

[BCS 89] British Computer Society and Institution of Electrical Engineers, "A Report on Undergraduate Curricula for Software Engineering", June 1989.

[Brooks 82] Brooks, Frederick., *The Mythical Man-Month : Essays on Software Engineering.*, Addison-Wesley, Reading, Massachusetts, (1975), Reprinted with corrections, (1982).

[Engle 89A] Engle, Charles, Ford, Gary, and Korson, Tim., *Software Maintenance Exercises for a Software Engineering Project Course.* Educational Materials CMU/SEI-89-EM-1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Feb. 1989.

[Engle 89B] Engle, Charles, Ford, Gary, and Tomayko, James., *APSE Interactive Monitor: A Software Artifact for Software Engineering Education.* Educational Materials CMU/SEI-89-EM-2, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Oct. 1989.

[Fagan 76] Fagan, M.E., "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, Number 3, 1976.

[Ford 90] Ford, Gary., "1990 SEI Report on Undergraduate Software Engineering Education", Technical Report CMU/SEI-90-TR-3, ESD-TR-90-204, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., March 1990.

[Mills 91] Mills, H.D., "Cleanroom Engineering: Engineering Software Under Statistical Quality Control," *American Programmer*, May 1991.

[Shaw 90] Shaw, Mary., *Prospects for an Engineering Discipline of Software* Technical Report CMU/SEI-90-TR-20, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Sep. 1990.

[Tomayko 87A]Tomayko, James., *Teaching a Project-Intensive Introduction to Software Engineering*. Technical Report CMU/SEI-87-TR-20, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Aug. 1987.

[Tomayko 87B]Tomayko, James., *Teaching a Project-Intensive Introduction to Software Engineering*. Special Report SEI-87-SR-1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Mar. 1987.